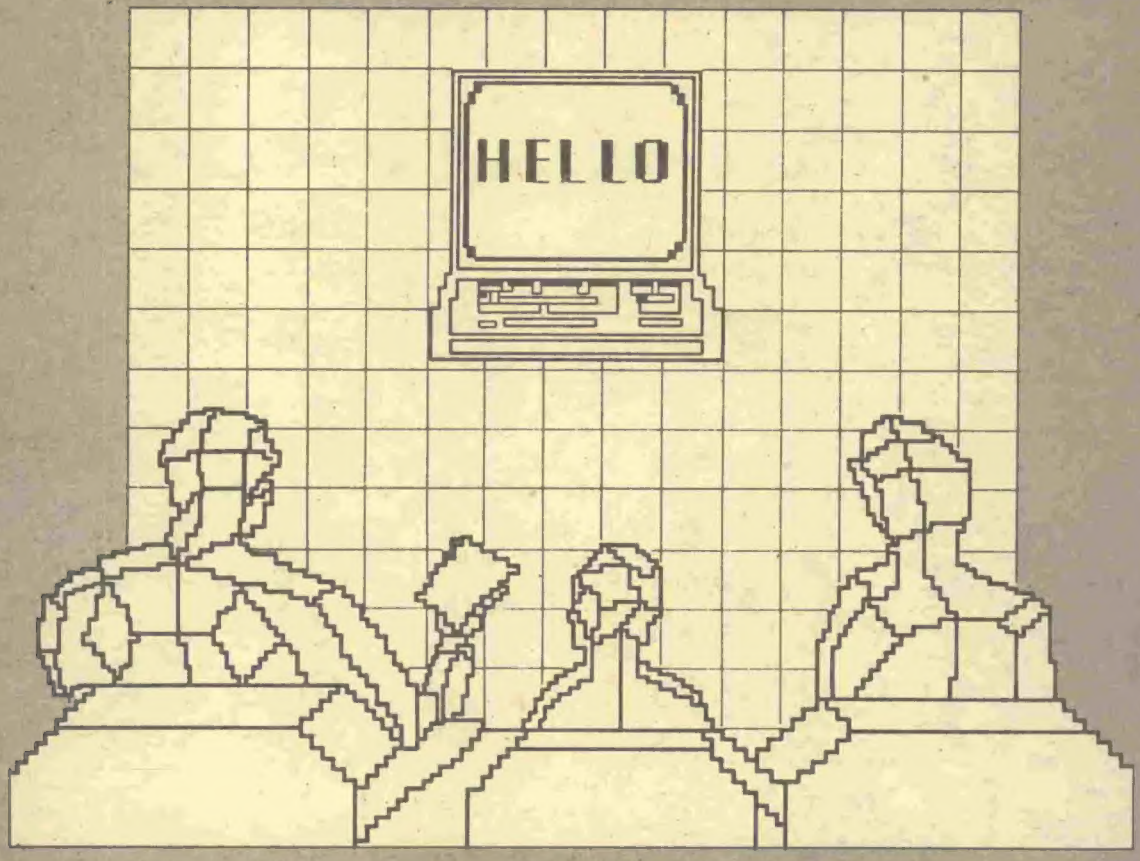
คอมพิวเตอร์ นักภาษาศาสตร์



บุญเลิศ เอี่ยมทัศนา

การสอนคอมพิวเตอร์ให้รู้ภาษาเป็นสาขาหนึ่งของปัญญาประดิษฐ์ (AI : Artificial Intelligence) ในภาษาอังกฤษชอบเรียกกันย่อ ๆว่า NLP (Natural-Language Processing) คือการทำให้คอมพิวเตอร์เข้าใจภาษามนุษย์และถือว่าเป็นสาขาที่สำคัญยิ่งสาขาหนึ่ง

ที่นักคอมพิวเตอร์ศาสตร์ให้ความสำคัญแก่มันมากก็เพราะว่าถ้าหากทำได้สำเร็จ ก็จะไม่ต้องเขียนโปรแกรมกันต่อไป เราเพียงแต่สั่งหรือบอกความต้องการของเราเครื่องก็จะทำให้ทันที และยิ่งผนวกการสังเคราะห์เสียงและภาพเข้าด้วย คอมพิวเตอร์ก็จะใกล้คน(ใช้) มากขึ้นทุกที งานนี้เป็นงานที่ยากมากแต่คงไม่นานเกินรอ ทราบมาว่าในญี่ปุ่นสามารถสั่งคอมพิวเตอร์ด้วยปากเปล่าให้วาดภาพได้แล้ว

สาขาของ NLP

ถึงแม้ความต้องการจะเหมือนกัน แต่การค้นคว้าวิจัยได้แยกเป็น 2 สาขา คือ

สาขาแรก ต้องการให้คอมพิวเตอร์เข้าใจภาษามนุษย์ทุกคำพูด เช่น สั่งว่า "วาดรูปเป็ดกำลังเดิน" คอมพิวเตอร์ก็

จะวาดรูปเป็ดซึ่งกำลังเดินอยู่ แต่ก็อาจจะถามกลับมาได้ว่าเป็ดสีอะไร? เดินอยู่บนอะไร? เป็นต้น สาขานี้ก็เพื่อให้คอมพิวเตอร์รู้จักสนทนาโต้ตอบกับมนุษย์ซึ่งอาจจะใช้ในการสั่งงานทั่วไป ตอบปัญหา ที่สนใจกันมากเป็นพิเศษคือการแปลภาษา เช่น ไทย เป็นอังกฤษ และกลับกัน

สาขาที่สอง ต้องการให้คอมพิวเตอร์รู้จักเฉพาะคำที่สำคัญในประโยคเท่านั้น เช่น สั่งว่า "แสดงรายชื่อลูกค้าที่เป็นหนี้เกิน ๕๐,๐๐๐ บาท" คำสำคัญจะมีเพียง แสดง หมายถึง ทางจอภาพ ชื่อลูกค้า หนี้ เกิน ๕๐๐๐๐ คำนอกจากนี้คอมพิวเตอร์จะไม่ใส่ใจ ทำให้สามารถสั่งคอมพิวเตอร์ได้โดยไม่ต้องใช้คำสั่งที่กำหนดเฉพาะ เช่นบางคนอาจจะสั่งว่า "กรุณาแสดงชื่อลูกค้าที่เป็นหนี้บริษัทเราอยู่เกินกว่า ๕๐,๐๐๐ บาท ทางจอภาพ" ก็จะได้รับการปฏิบัติที่เหมือนกัน

การเข้าใจภาษา

การให้คอมพิวเตอร์เข้าใจภาษามนุษย์ในทั้งสองสาขาดังกล่าวมานี้ โดยหลักการแล้วคล้ายกัน แตกต่างกันในรายละเอียดจึงขอนำสาขาแรกที่ให้คอมพิวเตอร์เข้าใจทุกคำพูดมากล่าวถึงซึ่งจะต้องดำเนินการเป็นหลายระดับ คือ

ระดับแรก จะต้องให้คอมพิวเตอร์รู้จักรูปแบบของประโยค (syntax) เช่น เป็นประโยคบอกเล่า (คำสั่ง) หรือเป็นประโยคคำถามและภายในประโยคจะต้องทราบด้วยว่า ใครเป็นประธาน ใครเป็นกรรม ทำอะไรกัน การที่จะให้คอมพิวเตอร์รู้เช่นนี้ จะทำได้โดยให้ตรวจสอบว่าเป็นประโยคที่ถูกต้องหรือไม่ ซึ่งทำได้โดยอาศัยหลักของวจีวิภาค (parsing) คือการกระจายคำ เช่นต้องตรวจสอบได้ว่า "สุนัขกินข้าว" เป็นประโยคที่ถูกต้อง

ระดับที่สอง จะต้องให้เข้าใจความหมายของประโยค (semantic) ซึ่งจะทำได้โดยให้ตรวจสอบว่าเป็นประโยคที่มีความหมายหรือไม่ เช่น ประโยค "สุนัขกินข้าว" กับ "ข้าวกินสุนัข" ทั้งสองประโยคถูกต้องตามหลักวจีวิภาค คือมีประธาน กริยา กรรม ทั้งคู่ แต่เมื่อมาทดสอบในระดับที่สอง จะต้องบอกได้ว่า "ข้าวกินสุนัข" เป็นประโยคที่ผิดความหมาย ในทำนองเดียวกันจะต้องบอกได้ว่า "สุนัขกินหิน" "กระต่ายเดิน" เป็นประโยคที่ผิดความหมาย นอกจากนี้ยังมีเรื่องอื่น ๆ อีกมาก เช่นจะต้องตรวจสอบได้ว่าประโยคในภาษาอังกฤษ "The bird is on the tree" หรือภาษาไทย "นกเกาะอยู่ในต้นไม้" เป็นประโยคที่ผิดความหมาย

ระดับที่สาม จะต้องให้คอมพิวเตอร์สนองตอบได้ว่า เมื่อมีประโยคคำถาม หรือคำสั่งใดแล้ว เมื่อเข้าใจแล้วจะดำเนินการอย่างไร

เท่าที่กล่าวมานี้ ไม่ใช่เรื่องง่าย แม้แต่ในระดับแรกเพียงระดับเดียวและกับภาษาเดียว เช่นจะให้คอมพิวเตอร์รู้จักรูปแบบของประโยคในภาษาอังกฤษทั้งหมดก็ไม่ใช่เรื่องง่ายแต่ก็มีผู้ดำเนินการค้นคว้าวิจัยกันอยู่ เพราะถ้าทำได้สำเร็จจะมีประโยชน์มหาศาล เช่นในการแปลภาษาระหว่างภาษามนุษย์ด้วยกัน (ไทย–อังกฤษ) จะทำได้ดีกว่า เพราะเป็นการแปลจากความเข้าใจของคอมพิวเตอร์ (ไทย→รูปแบบ→เข้าใจ→รูปแบบ→อังกฤษ) ไม่ใช่เป็นการแปลจากการเทียบคำ

วจีวิภาค (Parsing)

ในบทความนี้จึงขอนำเรื่องราวของการดำเนินการในระดับแรกคือ หลักการวจีวิภาคหรือการกระจายคำมากล่าวถึง แต่จะกล่าวเฉพาะประโยคในภาษาอังกฤษ (ภาษาไทยจะต้องมีเรื่องของการตัดคำมาเกี่ยวข้องอีก) ที่เป็นประโยคบอกเล่าและเป็นประโยคง่าย ๆ ที่ประกอบด้วย ประธาน กริยา กรรมเท่านั้น (กรรมจะมีหรือไม่ก็ได้) ซึ่งได้มีผู้กำหนดหลักการไว้ว่าให้แยกประโยคเช่นนี้เป็น NP (Noun Phrase) กับ VP (Verb Phrase) ซึ่งแต่ละอนุประโยคนี้จะประกอบด้วย

NP Pron
NP det + noun
NP det + adj + noun
NP prep + NP
VP verb
VP verb + NP
VP verb + adv
VP verb + adv + NP

ความหมายคือ NP อาจประกอบด้วย pronoun ตัวเดียวหรือ determiner คือคำนำหน้าคำนาม เช่น the กับคำนาม หรือ det กับ adjective กับคำนาม หรือ preposition กับ NP คือ ต่อด้วยแบบที่กล่าวมาทั้งหมดแบบใดแบบหนึ่ง ในเรื่องของ VP ก็เช่นกัน การกำหนดเช่นนี้ทำให้เขียนโปรแกรมได้ง่ายเรียกว่า production rules แต่ในที่นี้จะขอเรียกว่าไวยากรณ์เพื่ออ้างถึงต่อไปได้สะดวก

จะเห็นว่ายังไม่ครอบคลุมประโยคบอกเล่าธรรมดา เช่น เอกพจน์–พหูพจน์ อกรรมกริยา–สกรรมกริยา รวมทั้งเรื่องกาลด้วย แต่ก็เป็นพื้นฐานขั้นต้นได้เป็นอย่างดีครับ

วางแนวความคิด

เราสามารถมองโครงงานนี้ได้ ๒ นัย นัยแรกเป็นการมองตรง ๆ คือ สอนให้คอมพิวเตอร์รู้จักประโยคภาษาอังกฤษ ซึ่งจะทดสอบได้โดยป้อนประโยคภาษาอังกฤษทั้งถูกและผิดให้เครื่องตรวจสอบและแจ้งผล อีกนัยหนึ่ง มองเป็นการเขียนโปรแกรมเครื่องช่วยการเรียนการสอนภาษาอังกฤษ ซึ่งไม่ว่าจะมองในแบบใดจะได้โปรแกรมเดียวกันในบทความนี้จะมองในนัยหลังเพื่อแสดงถึงประโยชน์ใช้งานด้วย

เมื่อมองในแบบนี้ก็จะมองเห็นภาพคร่าว ๆ ว่าเราจะสอนให้คอมพิวเตอร์รู้จักไวยากรณ์และคำในภาษาอังกฤษ แล้วให้นักเรียนป้อนประโยคให้เครื่องตรวจสอบว่าถูกหรือผิด ถ้าผิดให้แสดงคำที่ผิดด้วย แต่การสอนให้เครื่องรู้จักไวยากรณ์และคำในภาษาอังกฤษนั้น จะให้เราผู้เขียนโปรแกรมเป็นผู้สอนหรือให้คุณครูภาษาเป็นผู้สอน ถ้าให้คุณครูเป็นผู้สอนได้จะดีมาก ซึ่งสำหรับไวยากรณ์เป็นเรื่องยากคงทำได้เฉพาะการสอนคำ เมื่อเป็นเช่นนี้จะแยกงานออกได้เป็น 3 เรื่องย่อยคือ

- สอนไวยากรณ์
- สอนคำ
- ตรวจสอบประโยค

งานสอนไวยากรณ์

ตามที่กล่าวแล้วว่าเราจะเป็นผู้สอนเอง แต่ก็สามารถพิจารณาต่อไปได้ว่าจะเขียนไว้ในโปรแกรมโดยตรงหรือให้ไว้เป็นข้อมูล ในขั้นแรกผมเขียนโปรแกรมนี้โดยใส่ไวยากรณ์ไว้ในโปรแกรมโดยตรงแต่เมื่อดำเนินการแล้วเห็นว่าไม่อ่อนตัว ไม่สามารถพัฒนาไปสู่แบบที่ให้คุณครูสอนได้ จึงเปลี่ยนมาเป็นเขียนไวยากรณ์ไว้เป็นข้อมูลดังแสดงในรูป B

ซึ่งเป็นการให้กฎของการกระจายคำซึ่งก็คือไวยากรณ์ แก่ตัวแปร nChr หมายถึง Noun Phrase และ vPhr หมายถึง Verb Phrase ในแบบการให้ค่าคงตัวแก่ตัวแปร (Type constant) ซึ่งจะอ่านได้เช่นเดียวกับที่กล่าวมาแล้ว คือ NP อาจประกอบด้วย pronoun เช่น He หรืออาจประกอบด้วย determiner และ noun เช่น The boy เป็นต้น

โดยมีแนวความคิดว่า จะอ่านประโยคที่นักเรียนป้อนเข้ามาตรวจสอบกับไวยากรณ์ ทีละคำว่าตรงกับคำใดเช่น ถ้าเป็น He ก็หยุด ถ้าเป็น The จะตรงกับกฎบรรทัดที่สองก็จะต้องตรวจต่อไปว่าคำต่อไปเป็น noun หรือไม่ เช่นถ้าเป็น boy ก็จะหยุด ถ้าไม่ใช่ก็จะตรวจบรรทัดที่สามว่าคำนี้เป็น adjective หรือไม่ เช่น ถ้าเป็น big ก็จะตรวจคำที่ถัดไปอีกว่าเป็น noun หรือไม่ และต่อ ๆ ไป

หากตรงกับเงื่อนไขที่กล่าวมาข้อใดข้อหนึ่งจะถือว่าประโยคนี้มี NP ถูกต้อง ก็จะตรวจ VP ต่อไป ในทำนองเดียวกัน แต่ถ้าตรวจมาถึงบรรทัดที่มีคำว่า ENDP จะเท่ากับไม่ตรงกับเงื่อนไขใดเลย แสดงว่าประโยคนั้นไม่มี NP และ VP ที่ถูกต้อง หากคุณสงสัยว่าเครื่องจะทราบได้อย่างไรว่า He เป็น pronoun ส่วน boy เป็น noun และคำอื่น ๆ อีก เป็นอีกเรื่องหนึ่งที่จะกล่าวในหัวข้อต่อไปครับ

งานสอนคำ

ดังได้กล่าวแล้วว่าจะให้คุณครูผู้สอนภาษาเป็นผู้สอนคำให้แก่คอมพิวเตอร์ด้วย ก็มีเรื่องที่จะต้องพิจารณาต่อไปว่า จะให้สอนเก็บไว้ในไฟล์ หรือสอนในขณะที่รันโปรแกรมนี้ การสอนในขณะที่รันโปรแกรมอาจเพิ่มความสนใจแก่นักเรียน ที่จะฝึกเขียนประโยคไปสอนคำแก่คอมพิวเตอร์ไปแต่จะทำให้โปรแกรมนี้ยาวเกินกว่าที่จะจบในตอนเดียว ผมจึงเลือกใช้วิธีเก็บไว้ในไฟล์

ไม่ว่าจะให้สอนในแบบใด จะต้องกำหนดรูปแบบของข้อมูลที่จะให้ป้อน การกำหนดนี้เราผู้เขียนโปรแกรมเป็นผู้กำหนดจะกำหนดอย่างไรก็ได้ แต่ควรเป็นแบบง่าย ๆ โดยพิจารณาถึงคำในภาษาอังกฤษด้วย เช่นคำว่า in เป็นได้ทั้ง adverb, adjective และ preposition จึงกำหนดให้ป้อนคำเป็น

in (adv, adj, prep) ;
boy (noun) ;

คือให้ขึ้นต้นด้วยคำ ตามด้วยชนิดของคำในวงเล็บซึ่งจะมีหนึ่งหรือหลายชนิดก็ได้ แต่ต้องไม่เกินสาม (หากกำหนดไว้มากก็เปลืองเนื้อที่ในหน่วยความจำ และคิดว่าคำในภาษาอังกฤษไม่น่าจะเป็นได้มากชนิดกว่านี้) สำหรับลำดับที่ในวงเล็บจะอยู่ก่อนหลังกันอย่างไรก็ได้

การป้อนคำจะใช้อิดิเตอร์ตัวใดก็ได้ เช่นของเทอร์โบปาสคาล สำหรับผมใช้ไซด์คิกป้อนไปทดสอบโปรแกรมนี้ไป ดังแสดงในรูปที่ 1 ซึ่งหากได้ครูสอนภาษา

```
{  Grammar Rules
   ==============  }

CONST
  nPhr : PhraseT =
    ((pron,FIN,non,non,non),
     (det,noun,FIN,non,non),
     (det,adj,noun,FIN,non),
     (det,adj,adj,noun,FIN),
     (prep,NP,FIN,non,non),
     (ENDP,non,non,non,non));
  vPhr : PhraseT =
    ((verb,adv,NP,FIN,non),
     (verb,adv,FIN,non,non),
     (verb,NP,adv,FIN,non),
     (verb,NP,FIN,non,non),
     (verb,FIN,non,non,non),
     (ENDP,non,non,non,non));
```

B

```
man(noun);  boy(noun);  girl(noun);
cat(noun);  rat(noun);  dog(noun);
rice(noun);  water(noun);  stone(noun);  cake(noun);

i(pron);  he(pron);  she(pron);  it(pron);  they(pron);

is(verb);  am(verb);  are(verb);
eat(verb);  drink(verb);  cry(verb);  bark(verb); smile(verb);
run(verb);

a(det);  an(det);  the(det);
Mr(det);  Miss(det,verb);

slowly(adv); quickly(adv);

on(prep,adv,adj);  in(prep,adv,adj);  under(prep,adv,adj);
to(prep,adv);

large(adj,adv);  black(adj);  fat(adj);
thin(adj);  red(adj);  small(adj);  little(adj);  big(adj);
```

รูปที่ 1 คำที่สอนให้คอมพิวเตอร์รู้จัก ในการทดลองรันโปรแกรม Grammar 3

กำหนดให้จะดีกว่า เสร็จแล้วให้เก็บไว้ในไฟล์ NEWBASE.TXT (ที่บังเอิญเป็นชื่อนี้เพราะผมปรับปรุงโปรแกรมใหม่ดังกล่าวแล้ว และก็เปลี่ยนรูปแบบของข้อมูลด้วยจึงให้ชื่อไว้เป็นที่สังเกต). ในการพัฒนาต่อไปอาจให้ผู้ใช้ป้อนชื่อไฟล์ข้อมูลก็ได้

การดำเนินการ

จากข้อมูลในไฟล์ดังกล่าว ผมมีแนวความคิดว่าจะโหลดลงหน่วยความจำและกำหนดเรียกเป็นฐานความรู้ (knowledge base) ซึ่งต้องขอให้ดูการกำหนดแบบของข้อมูลและตัวแปรในรูป A ก่อนครับ คือ

– ได้กำหนดขนาดของฐานความรู้ไว้เป็นค่าคงตัว BASESIZE เพียง 100 คำเท่านั้น สามารถจะขยายเพิ่มได้อีก

– สำหรับตัวฐานข้อมูลได้กำหนดไว้ในรูปอะเรย์ของเรคอร์ด KnowWrd เรียกว่า kBase (น่าจะกำหนดแบบของข้อมูลเป็นแบบทรี แทนที่จะเป็นอะเรย์ แต่จะทำให้โปรแกรมนี้ยาว)

– เรคอร์ด KnowWrd มีฟิลด์ word ซึ่งจะเก็บคำเช่น boy และฟิลด์ types ซึ่งจะเก็บชนิดของคำ เช่น noun แต่ได้ทำเป็นอะเรย์ขนาด 3 ช่อง เพื่อให้เก็บชนิดของคำได้ถึง 3 ชนิด ดังที่กล่าวมา

– การเก็บคำจะเก็บเป็นสตริงของคำนั้น ได้กำหนดไว้เป็นสตริงขนาด 20 อักขระ คือ tokeStrm (น่าจะเพียงพอ)

– การเก็บชนิดของคำ เพื่อความสะดวกในการเขียนโปรแกรม ได้เก็บไว้ในแบบที่กำหนดขึ้นใหม่ (user's define type) คือกำหนดเป็นแบบ Wrd Type

– และได้ทำตาราง WrdTab เพื่อเปลี่ยนแบบของข้อมูลไปมาระหว่างแบบที่เป็นสตริงกับแบบ WrdType

การดำเนินการสอนคำ จะเริ่มจากอ่านข้อมูลจากไฟล์ข้อมูล NEWBASE-TXT ซึ่งเป็นเท็กซ์ไฟล์เข้ามาทีละอักขระ ด้วยการอ่านข้ามอักขระที่เป็น วรรค วงเล็บเปิด วงเล็บปิด และ comma นำอักขระ A ถึง Z ซึ่งถูกเปลี่ยนเป็นอักษรใหญ่แล้วมารวมกันเป็นคำโดยใช้วรรค วงเล็บ และ comma สำหรับแยกคำ โปรแกรมที่ดำเนินการในเรื่องนี้อยู่ในรูป C และ D ซึ่งถือว่าคำแรก คือ คำ (พูด) ส่วนคำต่อไปคือชนิดของคำ ต่อเมื่อพบเครื่องหมาย ; จะถือว่าคำต่อไปเป็นคำแรกใหม่

ขอให้ดูในรูป D ครับ ตอนต้นเป็นการเปิดไฟล์ ถัดมาเป็นการเตรียมแฟลกต่าง ๆ เสร็จแล้วเข้าวงรอบ WHILE NOT EOFILE คือถ้ายังไม่สุดไฟล์ให้ดำเนินการต่อไปนี้ประโยคที่ต้องการเน้นเป็นพิเศษคือ

– WHILE แรก จะอ่านอักขระเข้ามาทีละตัวโดยเรียกใช้ Loadch (อยู่ในรูป C) ซึ่งเป็นฟังก์ชัน นำอักขระมาตรวจว่าเป็นพวกที่ต้องการข้ามไปหรือไม่ ขอให้ดูการกำหนดค่าคงตัว SKIP (ในรูป A) จะเห็นว่าเป็นพวก วรรค วงเล็บ และ comma ถ้าใช่ให้อ่านอักขระต่อไป เป็นการข้ามอักขระพวกนี้ ขอให้ดูท้ายประโยคที่มีคำว่า DO แล้วให้จบประโยค

A

```
{   This program will parse the English sentence which input
from the keyboard and show the error if any.
                              Boonlert Iamtasna
                                31 jan 31

PROGRAM   Grammar3(input,output,f1);
CONST     BASESIZE = 100;  MAXTOK = 20;
          MAXGRMR = 6;     MAXGRMC = 5;
          MAXT = 3;    { maximum number of type of word }
          SPACE = ' ';  STOP = '.';  CR = #13;
          OPEN  = '('; CLOSE = ')';  COMMA = ',';  SEMI = ';';
          SKIP  : SET OF Char = [SPACE,OPEN,CLOSE,COMMA];

TYPE      WrdType = (det,adj,noun,pron,verb,adv,prep,conj,intj,
                     FIN,NP,ENDP, non, none);
          Str4 = String[4];  Str20 = String[20];  Str80 = String[80];
          WordTabT = ARRAY[WrdType] OF Str4;
          KnowWrd = RECORD
             word : Str20;
             types : ARRAY[1..MAXT] OF WrdType;
                END;
          PhraseT = ARRAY[1..MAXGRMR,1..MAXGRMC] OF WrdType;
CONST     WrdTab : WordTabT = ('DET','ADJ','NOUN','PRON','VERB','ADV',
             'PREP','CONJ','INTJ','FIN','NP','ENDP','NON','NONE');

VAR       kBase : ARRAY[1..BASESIZE] OF KnowWrd;
          tokeStrm : ARRAY[1..MAXTOK] OF RECORD
                        word : Str20;
                        tt   : WrdType;
                          END;
          sent : Str80;
          maxNP,maxVP, kMax : Integer;
```

C

```
FUNCTION  CanLoadBase : Boolean;
VAR       Fi : Text;
          typ : WrdType;
          OK, frstWrd, EOFile : Boolean;
          ch : Char;     wrd : Str20;
          tIdx, i, t : Integer;

  FUNCTION  LoadCh(VAR ch : Char) : Char;
  BEGIN
    Read(Fi,ch);  ch := UpCase(ch);
    IF  ch = CR  THEN
      BEGIN  ch := SPACE;  ReadLn(Fi)  END;
    LoadCh := ch;
    IF  EOF(Fi) THEN  EOFile := TRUE;
  END;

  PROCEDURE  Tab(wrd : Str20; VAR w : WrdType);
  BEGIN
    w := det;
    WHILE  (wrd <> WrdTab[w]) AND (w <> none)  DO  w := Succ(w);
    IF  w = none  THEN
      WriteLn(wrd,' is not a word type');
  END;
```

เลย รูปแบบการเขียนโปรแกรมเช่นนี้ ผมนำมาจากภาษาซีครับ

– WHILE ต่อไปจะตรวจว่าอักขระเป็นอักษร A ถึง Z หรือไม่ ถ้าใช่จะนำมารวมกันใน Wrd โดยอ่านอักขระต่อไปจนกระทั่งไม่ใช่

– จากทั้งสอง WHILE นี้จะเหลืออักขระที่กำหนดให้ป้อนเป็นฐานข้อมูลเพียงตัวเดียวคือ ; ซึ่งนำมากำหนดเป็นเงื่อนไขแยกระหว่างคำ (พูด) กับชนิดของคำ ด้วยแฟลก frstWrd และดำเนินการกับ kmax และ t เท่ากับเป็นการเริ่มคำ (พูด) ใหม่

– ขั้นตอนของโปรแกรมที่เหลือเป็นไปดังที่กล่าวแล้ว คือ ถ้าเป็นคำ (พูด) ให้นำไปเก็บในฐานความรู้โดยตรง ถ้าเป็นชนิดของคำจะนำไปเทียบกับตารางโดยเรียกใช้ Tab (รูป C) ได้ชนิดของคำถามแบบข้อมูล WrdType นำไปเก็บในฐานความรู้

```
PROCEDURE  CheckEnd;
VAR   i : Integer;
BEGIN
  i := Ord(sent[0]);
  IF  sent[i] <> STOP  THEN
    BEGIN
    WriteLn('  <-- no period',#7);
    i := i + 1;  sent[i] := STOP;
    sent[0] := Chr(i);
    END;
END;
```

E

งานตรวจสอบประโยค

เป็นอันว่าเราได้สอนคอมพิวเตอร์ให้รู้จักไวยากรณ์และคำในภาษาอังกฤษแล้ว ขั้นต่อไปคือการนำมาใช้งานซึ่งจะให้ตรวจสอบประโยคที่นักเรียนป้อนว่าถูกหรือผิด โปรแกรมในส่วนนี้คือเป็นโปรแกรมที่จะประสานความรู้ทั้งหลายของคอมพิวเตอร์เข้าด้วยกัน ซึ่งแบ่งเป็นงานย่อยได้อีก 3 งานคือ

การตรวจท้ายประโยค

ประโยคบอกเล่าในภาษาอังกฤษจะต้องปิดท้ายประโยคด้วยจุด (period) ดังนั้นการดำเนินการในลำดับแรกจึงควรจะตรวจสอบว่านักเรียนปิดท้ายประโยคด้วยจุดหรือไม่ เพื่อนำไปเป็นเงื่อนไขในการเขียนโปรแกรมด้วย ปัญหาที่ตามมาคือ ถ้าไม่ได้ใส่จุดใช้จะให้ทำอย่างไร จะให้ผิดเลยหรือช่วยเติมจุดให้ผมเลือกเอาอย่างหลังแต่ให้แสดงคำเตือนพร้อมทั้งส่งเสียงปี๊บ 1 ครั้ง (เป็นการลงโทษ)

การตรวจท้ายประโยคก็เพียงแต่ตรวจดูว่าท้ายสตริงที่นักเรียนป้อนมีจุดหรือไม่ ดังแสดงในโปรแกรม Check End รูป E

```
BEGIN  { CanLoadBase }
  Assign(Fi, 'NEWBASE.TXT');
  {$I-}  Reset(Fi);  {$I+}
  OK := (IOresult = 0);  CanLOadBase := OK;
  IF  NOT OK  THEN
    WriteLn('Can not open file KNOWBASE.TXT')
  ELSE
    BEGIN
    frstWrd := TRUE;  EOFile := FALSE;  kMax := 1;  t := 1;
    WHILE  NOT EOFile  DO
      BEGIN
      i := 1;
      WHILE  (LoadCh(ch) IN SKIP) AND (NOT EOFile)  DO ;
      WHILE  (ch IN ['A'..'Z']) AND (NOT EOFile)  DO
        BEGIN  wrd[i] := ch;  i := i + 1;
               ch := LoadCh(ch)                END;
      wrd[0] := Chr(i - 1);
      IF  ch = ';'  THEN
        BEGIN  frstWrd := TRUE; kMax := kMax + 1; t := 1  END
      ELSE IF  frstWrd  THEN
        BEGIN  kBase[kMax].word := wrd;  frstWrd := FALSE  END
      ELSE
        BEGIN  Tab(wrd,typ);
               kBase[kMax].types[t] := typ;  t := t + 1  END;
      END;
    kBase[kMax].word := 'nowrd';  kBase[kMax].types[1] := none;
    END;
  Close(Fi);
END;
```

D

งานแยกคำ

ประโยคที่นักเรียนป้อนผมให้เก็บไว้ในตัวแปรสตริง sent เช่นนักเรียนป้อนประโยคว่า

The dog run.

จะได้ข้อมูลประโยคนี้ในตัวแปร sent ซึ่งคอมพิวเตอร์จะมองเห็นเป็นแถวอักขระ T,h,e, วรรค, d,o,g, วรรค, r,u,n และจุด

แต่เราต้องการให้คอมพิวเตอร์เห็นเป็นคำ 3 คำ คือ The, dog และ run กับจุด ดังนั้นเราจะต้องแยกคำเหล่านี้ออกจากประโยค ซึ่งทำได้ 2 แบบ คือ แยกออกมาทีละคำแล้วนำไปตรวจสอบ ถ้าผิดให้ยุติการดำเนินการต่อไป (ดังที่ใช้ในเรื่อง ภาษา IQO) อีกแบบหนึ่งคือแยกคำทั้งหมดเก็บไว้แล้วจึงนำไปตรวจสอบ

ในกรณีนี้การแยกคำทั้งหมดก่อนจะทำให้การทำงานของโปรแกรมโดยส่วนรวมใช้เวลาน้อยกว่า เพราะเราต้องเทียบคำลึกเข้าไปในประโยคหลายครั้ง และเมื่อผิดต้องมาเริ่มที่คำแรกใหม่ การแยกคำทีละคำไม่ได้เก็บคำไว้ ดังนั้น การเริ่มที่คำแรกก็คือการเริ่มที่อักขระตัวแรก (น่าจะใช้วิธีผสมคือแยกทีละคำและพร้อมกันนี้ให้เก็บคำที่แยกแล้วไว้ด้วย แต่ก็มาคิดได้ภายหลังครับ)

การแยกคำคือการรวมอักขระที่เป็นอักษร A ถึง Z ที่อยู่ติดกันเข้าด้วยกัน คำที่แยกแล้วผมกำหนดเรียกเป็นโทเค็น (token) เพื่อสะดวกในการอ้างถึง โปรแกรมที่ดำเนินการในเรื่องนี้คือโพรซีเดอร์ TokenSentence ในรูป F

ตัวโปรแกรมหลักของมันอยู่ตอนล่าง ดำเนินการโดยเรียกใช้ Form-Token (รูปเดียวกัน) ให้แยกคำมาทีละโทเค็นแล้วนำไปเก็บในอะเรย์ takeStrm จนกว่าจะจบประโยค คือได้โทเค็นเป็นจุด

สำหรับโพรซีเดอร์ FormToken จะอ่านคำจากสตริง sent ทีละอักขระ

โดยเรียกใช้ Getch ซึ่งเพียงแต่นำอักขระในสตริงมาแปลงเป็นอักษรใหญ่ให้แก่ตัวแปร ch ทีละตัว ขอให้สังเกตด้วยครับว่าผมให้ Getch เป็นฟังก์ชันเพื่อใช้ในวงรอบ WHILE (ตามแบบภาษาซี)

ใน FormToken วงรอบ WHILE แรกเป็นการอ่านข้ามวรรค ถ้ามี วงรอบ WHILE ต่อไปเป็นการรวมตัวอักษรเก็บตรง ๆ ในตัวแปร wrd จะได้โทเค็น นี้เป็นจุดแต่เพียงตัวเดียว แต่ถ้านักเรียนป้อนอักขระอื่นอาจเป็น , หรือตัวเลข ก็จะถูกเก็บใน wrd เป็นโทเค็นละตัวเช่นกัน ส่งไปตรวจสอบต่อไป

งานตรวจสอบ

ขณะนี้คอมพิวเตอร์รู้จัก ไวยากรณ์แล้ว รู้จักคำในฐานะชนิดของคำแล้ว และรู้จักประโยคที่นักเรียนป้อนเป็นโทเค็นแล้ว งานในขั้นตอนต่อไปคือการนำมาตรวจสอบ ซึ่งอาจทำได้หลายแบบที่ผมรู้จัก คือ State-Machine NLP Parser กับ Context-Free NLP Parser แบบแรกมีขีดจำกัดมาก แบบที่สองอ่อนตัวกว่าและที่เตรียมการมาทั้งหมดนี้ก็เพื่อการดำเนินการในแบบที่สอง แต่ผมได้ดัดแปลงไปจากต้นแบบเพื่อให้อ่อนตัวยิ่งขึ้น

โปรแกรมที่ดำเนินการในเรื่องนี้คือ ParseSentence มีโปรแกรมหลักอยู่ในรูป J ซึ่งมีการดำเนินการโดยย่อดังนี้คือ ParseSentence จะกระจายประโยคโดยถือว่าในแต่ละประโยคจะต้องประกอบด้วยวลี NP ตามด้วย VP ในแต่ละวลีนี้จะประกอบด้วยคำ จึงนำคำในไวยากรณ์ (typ : type) กับคำในประโยค (toke : token) มาเทียบกัน

เทียบคำ

ผมจึงขอนำเรื่องของการเทียบคำซึ่งมอบให้โพรซีเดอร์ CompToken รูป G มากล่าวถึงก่อน การเทียบคำจะเริ่มต้นหากเป็นอักษร เสร็จแล้วจะมาถึงเงื่อนไข IF จะเห็นว่าผมได้นำค่าในตัวแปร wptr ซึ่งเป็นอินเด็กซ์สำหรับสตริง wrd มาเป็นเงื่อนไขว่าได้อ่านโทเค็นคือรวมอักษรมาหรือไม่

– ถ้าอ่านโทเค็นมาค่าใน wptr จะมากกว่าศูนย์นำไปให้เป็นความยาวของโทเค็นที่สตริง wrd ช่องศูนย์และลดหนึ่งค่า sptr ซึ่งเป็นอินเด็กซ์สำหรับสตริง sent (ในกรณีนี้ดำเนินการอย่างอะเรย์)เพราะ Getch จะทำให้เกินไปหนึ่ง

– แต่ถ้าไม่ได้อ่านโทเค็นคือ ไม่ได้ทำงานในวงรอบ WHILE ที่สอง wptr จะมีค่าเป็นศูนย์ตามที่ให้ค่าไว้แต่แรก ดังนั้น ค่าใน ch จะเป็นอักขระที่ไม่ใช่อักษรและวรรค ที่เราหวังไว้คือจุดนำมา

F

```
PROCEDURE  TokenSentence;
VAR   sptr, tptr : Integer;
      ch : Char;  word : Str20;

  PROCEDURE  FormToken(VAR wrd : Str20);
  VAR  wptr : Integer;

    FUNCTION  GetCh(VAR ch : Char) : Char;
    BEGIN
      ch := UpCase(sent[sptr]);  GetCh := ch;
      sptr := sptr + 1;
    END;

  BEGIN  { FormToken }
    wptr := 0;
    WHILE  GetCh(ch) = SPACE  DO  { skip space } ;
    WHILE  ch In ['A'..'Z']  DO
      BEGIN  wptr := wptr + 1; wrd[wptr] := ch;
             ch := GetCh(ch);                    END;
    IF  wptr > 0  THEN
      BEGIN  wrd[0] := Chr(wptr);  sptr := sptr - 1  END
    ELSE
      BEGIN  wrd[1] := ch;  wrd[0] := Chr(1)  END;
  END;

BEGIN  { TokenSentence }
  sptr := 1;  tptr := 1;
  REPEAT
    FormToken(word);
    tokeStrm[tptr].word := word;  tptr := tptr + 1;
  UNTIL  word = STOP;
END;
```

G

```
PROCEDURE   ParseSentence;
VAR     tIdx, lastI, errata, start : Integer;

PROCEDURE  CompToken(tpe : WrdType; toke : Str20);
VAR    kidx, i : Integer;

{###}
procedure  tracking;
begin
  writeln;
  write('|',wrdtab[tpe],'|',toke);
  write('(',kidx,')','(',i,')','(',tIdx,')');
  write(' err=',errata);
end;

BEGIN   { CompToken }
  toke := tokeStrm[tIdx].word;  kidx := 1;
  WHILE  (toke <> kBase[kidx].word) AND (kidx < kMax)  DO
    kidx := kidx + 1;
  IF  kidx = Kmax  THEN  errata := 1     { don't know this word }
  ELSE
    BEGIN
    i := 1;
    WHILE  (tpe <> kBase[kidx].types[i]) AND (i <= MAXT)
      DO  i := i + 1;
    IF  i > MAXT  THEN  errata := 2       { miss match }
    ELSE
      errata := 0;
    END;
  IF  tIdx > lastI  THEN
    BEGIN  lastI := tIdx;  tokeStrm[lastI].tt := tpe  END;
{##}   tracking;
END;
```

ด้วยการนำ toke ไปตรวจหาในฐานความรู้ kBase ว่าคอมพิวเตอร์รู้จักคำนี้หรือไม่ถ้าไม่รู้จักจะให้ error = 1 ซึ่งจะใช้เป็นเงื่อนไขในการยุติการตรวจเทียบต่อไป

หากพบคำนี้ในฐานความรู้ก็จะนำมาตรวจเทียบกับ tpe ว่าเป็นคำชนิดเดียวกัน (noun, verb, adj ฯลฯ) หรือไม่ แต่ชนิดของคำในฐานะความรู้อาจมีมากกว่าหนึ่งจึงนำ tpe ไปตรวจเทียบ ถ้าตรงกันจะให้ error = 0 เพื่อนำไปเป็นเงื่อนไขในการตรวจเทียบรูปแบบของวลีนั้นต่อไป หากไม่ตรงกันจะให้ error = 2 เพื่อแสดงว่าประโยคนี้ไม่ตรงกับแบบของวลีที่กำลังตรวจอยู่ นำไปเป็นเงื่อนไขให้ตรวจวลีอื่นต่อไป ขอสรุปอีกครั้งว่า

- error = 1 ไม่มีคำนี้ในฐานความรู้ ให้ยุติการตรวจประโยค
- error = 0 คำตรงกับไวยากรณ์ให้ตรวจวลีนั้นต่อไป
- error = 2 คำไม่ตรงกับไวยากรณ์ ให้ตรวจกับวลีอื่น

สำหรับตอนท้ายที่ให้เก็บค่า tidx ไว้จะขอนำไปกล่าวในเรื่องการแสดงคำที่ผิดครับ

เทียบวลี

เพราะผมได้กล่าวถึงการเทียบคำก่อนการอธิบายของผมจึงเป็นในแบบถอยหลัง (ขอได้โปรดทราบตามนี้ด้วย) ขอแนะนำตัวแปรที่จะนำมาใช้ก่อน สองตัวแรกขอให้ดูตัวแปร nPhr และ vPhr ซึ่งได้ให้ค่าคงตัวไว้ในรูป B ตัวแปรทั้งสองนี้เป็นอะเรย์สองมิติ ผมจะกำหนดเรียกอินเด็กซ์เป็น row และ col ทั้งคู่ คือในแต่ละอะเรย์จะมี 6 row กับ 5 col

row หนึ่ง ๆ คือแบบของวลีหนึ่ง ๆ (เมื่อไม่สับสนจะใช้คำว่าวลีเพียงคำเดียว) ส่วน col หนึ่ง ๆ คือ type หนึ่ง จากการตรวจเทียบคำจะได้ผลในตัวแปร error ซึ่งผมมีแนวความคิดพร้อมทั้งจะยกตัวอย่างประกอบดังนี้ เช่นนักเรียนป้อนประโยคว่า

The black cat cry.

และสมมติว่าเราได้สอนให้เครื่องรู้จักว่า the เป็น det, black เป็น adj และ cat เป็น noun ส่วน cry ไม่ได้สอนไว้ เมื่อเริ่มงานตรวจเทียบประโยคจะเริ่มด้วยการตรวจ NP ก่อน คำแรกที่นำมาเทียบกันคือ pron กับ the (ขอให้ดูรูป B ประกอบ) ได้ผลของการเทียบคำใน error = 2 เพราะมี the ในฐานข้อมูลแต่ชนิดของคำไม่ตรงกับวลีที่กำลังตรวจอยู่

จะนำ error มาเป็นเงื่อนไขให้ตรวจเทียบวลีต่อไปคือ row = 2 เทียบ det กับ the ตรงกันได้ error = 0 ให้ตรวจในวลีเดียวกันนี้ต่อไป เทียบ noun กับ black ได้ error = 2 ให้ตรวจวลีต่อไปคือ row = 3 เริ่มเทียบ det กับ the อีกตรงกัน เทียบ adj กับ black ตรงกัน ตรวจ noun กับ cat ตรงกันอีกได้ error = 0 ให้เทียบในวลีเดียวกันนี้ต่อไปอีก คือเทียบ FIN กับ Cry

แต่คำว่า FIN กำหนดไว้ว่าจบแบบของวลีหนึ่ง ๆ จึงไม่ให้ตรวจต่อไป ดังนั้นในการเทียบวลีจึงประกันแต่เพียงว่าในประโยคที่ให้ตรวจนั้นมีคำครบถ้วนในแต่ละวลีตามที่กำหนดในไวยากรณ์แต่ไม่ประกันว่าจะมีเกินหรือไม่ ที่ออกแบบใช้เช่นนี้เพื่อให้เขียนโปรแกรมง่ายและเพื่อการพัฒนาต่อไป

หลังการตรวจ NP ว่าถูกต้องแล้วจะให้ตรวจ VP ต่อไป ซึ่งจะดำเนินการเช่นกัน โดยเทียบ verb กับ cry ปรากฏว่าเราไม่ได้สอนคำว่า cry ไว้ได้ error = 1 ให้ยุติการตรวจเทียบไปรายงานผลเลย แต่ถ้าสอนไว้ว่า cry เป็น verb จะดำเนินการตรวจเทียบไปจนถึง row = 4 คือในวลีของ vPhr ที่มี verb เพียงคำเดียวได้ผลการตรวจสอบว่าถูกต้อง

โปรแกรมที่ดำเนินการในเรื่องนี้คือ TestPhrase ในรูป H ซึ่งเป็นโปรแกรมย่อยให้ TestNP กับ TestVP เรียกใช้ ขอให้สังเกตบนส่วนหัวของ TestPhrase ว่า มีการกำหนด TestNP ไว้เป็นชื่ออ้างถึงล่วงหน้า (Forward reference : ซึ่งหาตัวอย่างใช้งานจริง ๆ ได้ยากพอสมควร) ทั้งนี้เพราะโพรซีเดอร์ TestNP เป็นโปรแกรมเรียนตนเอง (ตาม

H

```
FUNCTION  TestNp : Boolean; Forward;

PROCEDURE  TestPhrase(ph : PhraseT; row : Integer);

  PROCEDURE  TestRow(ph : PhraseT; r,c : Integer);
  VAR  tpe : WrdType;  toke : Str20;
       OK : Boolean;   tem : Integer;
  BEGIN
    tpe := ph[r,c];  toke := tokeStrm[tidx].word;
    IF  (tpe <>FIN) AND (toke <> STOP) AND (errata = 0)  THEN
      BEGIN
      IF  tpe = NP  THEN
        BEGIN  OK := TestNP;  c := c + 1  END
      ELSE
        BEGIN
        CompToken(tpe,toke);
        IF  errata = 0  THEN
          BEGIN  c := c + 1;  tidx := tidx + 1  END;
        END;
      TestRow(ph,r,c);
      END;
  END;

BEGIN  { TestPhrase }
  IF  (errata > 1) AND (ph[row,1] <> ENDP)  THEN
    BEGIN
    errata := 0;
    TestRow(ph,row,1);
    IF  errata > 1  THEN  BEGIN  row := row + 1; tidx := start  END;
    TestPhrase(ph,row);
    END;
END;
```

แบบของข้อมูลคือไวยากรณ์) แต่การเรียกตนเองไปทำในโปรแกรมย่อย TestPhrase

ขอให้ดูในรูป B อีกครั้งครับ ใน nPhr ทำให้ TestNP เรียกตนเองซึ่งก็มีเพียงแห่งเดียวเท่านั้น คือใน row = 5 ที่มีคำว่า NP อยู่ ส่วนใน vPhr ที่มีคำว่า NP ไม่ได้ก่อให้เกิดการเรียกตนเอง เป็นเพียงทำให้มีการเรียกใช้ TesNP ตรง ๆ เท่านั้น แต่เนื่องจากผมให้ใช้ TestPhrase ร่วมกันจึงกลายเป็นการเรียกตนเองไป

การทำงานของโปรแกรม Test Phrase ตอนล่างของรูป H เป็นไปตามที่กล่าวมา โดยตัวของมันเองจะตรวจประโยคโดยเทียบกับไวยากรณ์ทีละวลี จนกว่าจะตรงกัน (error = 0) หรือไม่มีคำนั้นในฐานความรู้ (error = 1) หรือหมดแบบของวลีที่จะให้เทียบ (ENDP) แต่เนื่องจากผมเขียนเป็นโปรแกรมเรียกตนเอง จึงต้องกำหนดเงื่อนไขไว้ดังที่เห็นในโปรแกรม

ในการตรวจเทียบวลีได้เขียนเป็นโปรแกรมย่อย TestRow โดยจะเทียบคำ (token) ในประโยคกับไวยากรณ์ในวลีหนึ่งคือ row หนึ่ง จนกว่าแบบของคำไม่ตรงกัน (error = 2 หรือ 1) หรือสุดวลี (FIN) หรือสุดประโยค (STOP) แต่เขียนเป็นโปรแกรมเรียกตนเองจึงต้องกำหนดเงื่อนไขดังที่เห็นในโปรแกรมอีกเช่นกัน

ในโปรแกรม TestRow มีการเรียกตนเองอยู่ 2 แบบ คือที่เรียกตัวของมันเองแท้ ๆ คือ Test Row (อยู่ตอนล่างของโปรแกรม) เพื่อจะตรวจคำไปตาม row ทีละคำจนกว่าจะเป็นไปตามเงื่อนไขที่กล่าวมา อีกแบบหนึ่งคือเมื่อ tpe = NP จะเรียก TestNP เพื่อให้เตรียมค่าต่างๆ ให้ซึ่งสุดท้ายจะเป็นการเรียกตัวมันเอง (indirect recursion)

ที่ผมเขียนเป็นโปรแกรมเรียกตนเองใช้หลายแห่งหลายรูปแบบก็เพื่อแสดงวิธีการเรียกตนเองในแบบต่าง ๆ เท่านั้น ความจริงแล้วใช้วงรอบ WHILE หรือ UNTIL แทนได้ ที่ต้องการการเรียกตนเองแท้ ๆ มีเพียงแห่งเดียวเท่านั้นคือตรงที่ตรวจสอบ typ = NP แล้วเรียกใช้ TestNP (ซึ่งเป็นการเรียกตนเอง) ทั้งนี้เพราะว่า

ก่อนที่จะเรียก TestNP โปรแกรมอาจทำงานอยู่กับ nPhr หรือ vPhr สมมติว่าทำงานอยู่กับ vPhr เมื่อเรียก TestNP จะเป็นการทำงานกับ nPhr เมื่อเสร็จสิ้นการทำงานแล้วส่งกลับ (return) เพราะเป็นการเรียกตนเองจึงทำให้โปรแกรมกลับไปทำงานกับ vPhr ตามเดิม และคืนค่า c เดิมมาให้ด้วย ทำให้เราสามารถตรวจในวลีของ vPhr เดิมต่อไปได้ คือสามารถตรวจประโยคที่มีวลี VP ประกอบด้วย verb, NP และ adv ได้

ตรวจวลี NP และ VP

โปรแกรมสำหรับตรวจว่าส่วนของประโยคนั้นเป็นวลี NP และ VP ตามที่กำหนดไว้ในไวยากรณ์หรือไม่อยู่ในรูป I มีการทำงานเหมือนกัน คือเริ่มต้นด้วยการเก็บค่า tidx ซึ่งเป็นอินเด็กซ์สำหรับชี้ประโยคที่นักเรียนป้อนไว้ในตัวแปร start เพื่อให้ TestPhrase ใช้เมื่อขึ้น row คือวลีใหม่ ที่ให้ error = 9 เพื่อบังคับให้เข้าทำงานในโปรแกรม Test Phrase

ตรวจประโยค

คือโปรแกรม ParsSentence ที่ได้กล่าวถึงมาบ้างแล้ว ตัวโปรแกรมหลักอยู่ในรูป J จะดำเนินการตามที่กล่าวมาแล้วคือตรวจว่ามีวลี NP หรือไม่ ถ้ามีให้ตรวจต่อไปว่ามีวลี VP หรือไม่ ถ้ามีให้ตรวจว่าจบประโยคหรือไม่ (ถือมีคำเกินมาหรือไม่) ถ้าถูกต้องก็จะให้ข้อความ

```
FUNCTION  TestNp;
BEGIN
  start := tidx; errata := 9;
  TestPhrase(nPhr,1);
  TestNp := (errata = 0);
END;

FUNCTION   TestVP : Boolean;
BEGIN
  start := tidx; errata := 9;
  TestPhrase(vPhr,1);
  TestVP := (errata = 0);
END;
```

I

```
PROCEDURE  DisplayErrMessage;

  procedure  FirstError;
  begin WriteLn('pronoun or determiner or preposition')  END;

BEGIN
  IF  errata <> 0  THEN
    BEGIN
    Write(tokeStrm[lastI].word,' <-- ');
    IF  errata = 1  THEN  Write('I don''t know this word ')
    ELSE  Write('This word ');
    Write('suppose to be ');
    CASE  tokeStrm[lastI].tt  OF
     pron : FirstError;
     noun,adj : WriteLn('noun or adjective');
     verb : WriteLn('verb');
     adv  : begin WriteLn('adverb or '); FirstError  end;
     NP   : FirstError;
    END;  { case }
    END;
END;


BEGIN  { ParseSentence }
  tIdx := 1; lastI := 0;
  IF  TestNP = TRUE  THEN
    IF  TestVP = TRUE  THEN
      IF  tokeStrm[tIdx].word = STOP  THEN
        WriteLn(' is a correct sentence')
      ELSE
        WriteLn(' sentence is too long')
    ELSE
      WriteLn(' error in Verb Phase')
  ELSE
    WriteLn(' error in Noun Phase');
  DisplayErrMessage;
END;
```

J

ว่าถูกต้อง ถ้าไม่เป็นตามเงื่อนไขที่กล่าวมาก็จะแสดงข้อความว่าผิดตรงใด ดังที่เห็นในโปรแกรม

สำหรับตัวแปร lastI ที่ให้ค่าเป็นศูนย์แต่ต้น ก็เพื่อเก็บค่าของอินเด็กซ์ tIdx ที่มากที่สุด (ดำเนินการอยู่ในโปรแกรม CompToken อยู่ตอนท้ายของโปรแกรม) คือคำในประโยคที่ลึกที่สุดที่ตรวจสอบไปถึง และได้เก็บชนิดของคำไว้ด้วย เพื่อนำมาใช้ในโปรแกรม Display ErrMessage อยู่ในรูป J เช่นกัน ซึ่งจะแจ้งให้ผู้ใช้คือนักเรียนทราบว่าประโยคนั้นผิดตรงคำไหน เมื่อมีที่ผิด

โปรแกรมหลัก

ตัวโปรแกรมหลักของโครงงานนี้อยู่ในรูป L เริ่มด้วยการเคลียร์ค่าในฐานความรู้ด้วย ClarBase (ตัวโปรแกรมอยู่ในรูป K) เสร็จแล้วให้โหลดข้อมูลจากไฟล์ลงฐานความรู้ kBase ด้วยโปรแกรม CanLoadBase ซึ่งถ้าโหลดได้สำเร็จจึงจะดำเนินการต่อไป ถ้าโหลดไม่ได้จะแจ้งให้ทราบว่าโหลดไม่ได้

หากโหลดได้ก็จะได้แสดงข้อมูลในฐานความรู้ในครั้งแรก เพียงครั้งเดียวด้วยโพรซีเดอร์ dispbase ที่ทำเครื่องหมาย ## ไว้เพื่อแสดงว่าบรรทัดนี้ตัดทิ้งได้ หลังจากนี้จะอ่านประโยคที่นักเรียนป้อนมาไว้ในตัวแปร sent ถ้าไม่ใช่คำว่า 'quit' ให้ตรวจว่าปิดท้ายประโยคด้วยจุดหรือไม่ด้วยโพรซีเดอร์ CheekEnd ดังที่กล่าวมาแล้ว

เสร็จแล้วให้แยกคำใน sent เป็นโทเค็นใน tokeStrm ด้วยโพรซีเดอร์ TokenSentence สำหรับ disptoke ที่ใส่ไว้เป็นหมายเหตุ เผื่อคุณผู้อ่านต้องการเห็นข้อมูลที่แยกคำแล้ว ให้ถอดวงเล็บปีกกาออก จากนี้ไปก็จะดำเนินการกระจายคำด้วยโพรซีเดอร์ PharseSentence ดังที่กล่าวมาแล้ว แล้วกลับไปให้ป้อนประโยคใหม่ต่อไป ทำซ้ำอยู่เช่นนี้จนกว่าจะเป็นคำว่า "quit"

โปรแกรมช่วย

ในรูป G ได้สร้างโพรซีเดอร์ tracking ไว้เป็นโปรแกรมช่วย ทั้งในด้านการพัฒนาและศึกษาการทำงานของโปรแกรมนี้ ซึ่งจะแสดงชนิดของคำตามที่กำหนดไว้ในไวยากรณ์คือใน nPhr และ vPhr กับคำในประโยคที่ให้ตรวจสอบค่าใน tidx และผลของการตรวจสอบคือค่าใน error การใช้ tracking อยู่ท้ายโปรแกรม CompToken ในรูปเดียวกันนี้ หากไม่ต้องการให้ตัดออกได้ครับ

สิ่งที่น่าสนใจ

โปรแกรมในโครงงานนี้ได้แสดงการเขียนโปรแกรมในภาษาปาสคาลที่หาตัวอย่างมาแสดงได้ยาก คือ

1. การให้ค่าคงตัวแก่ตัวแปร (Type Constant)

– แบบเซต คือ SKIP

K

```
procedure  disptoke;
var  i : Integer;  str : str20;
begin
  i := 1;
  repeat
    str := tokestrm[i].word; i := i + 1;
    write(str,SPACE);
  until  str = STOP
end;

procedure  dispbase;
var  i : Integer; wrdt : str4; tke : wrdtype;
begin
  for i := 1 to kmax  do
    begin
    tke := kbase[i].types[1]; wrdt := WrdTab[tke];
    write(kbase[i].word,SPACE,wrdt,'; ');
    end;
end;
(*.....................................................................*)
PROCEDURE  ClearBase;
CONST  datum : KnowWrd =
           (word : 'nowrd';
           types : (none,none,none));
VAR   i : Integer;
BEGIN
  FOR i := 1 TO BASESIZE  DO
    kBase[i] := datum;
END;
```

L

```
var frst : boolean;

BEGIN  { MAIN }
frst := TRUE;
ClearBase;
IF  CanLoadBase  THEN
  REPEAT
  IF FRST THEN  BEGIN  FRST := FALSE;  dispbase;  END;  {##}
  WriteLn; Write(': '); Read(sent);
  IF  sent <> 'quit'  THEN
    BEGIN
    CheckEnd;
    TokenSentence;
{   disptoke;}
    ParseSentence;
    END;
  UNTIL  sent = 'quit'
ELSE
  WriteLn(' Can not open file  NEWBASE.TXT');
END.
```

– อะเรย์มิติเดียวคือ WrdTab (แบบนี้ใช้บ่อยเหมือนกันนำมากล่าวเพื่อให้เข้าชุดกัน)

– อะเรย์สองมิติ คือ nPhr และ vPhr

– เรคอร์ด คือ datum (รูป K) ซึ่งเป็นเรคอร์ดที่มีฟิลด์เป็นอะเรย์ด้วย

2. การใช้ชื่ออ้างถึงล่วงหน้า (forward reference) คือฟังก์ชัน TestNP ในรูป H และ I